FACULTY
OF INFORMATICS
Masaryk University

PA039: Supercomputer Architecture and Intensive Computing

Parallel computers

Luděk Matyska

Spring 2023

Parallel computers

- Small-scale multiprocessing
  - 2–several hundreds of cores (ten of processors)
  - mostly SMP (shared memory systems)
- Large-scale multiprocessing
  - from hundreds to millions of cores (processors)
  - Most often distributed memory

Parallel computers (II)

- Architecture
  - Single Instruction Multiple Data, SIMD
  - Multiple Instruction Multiple Data, MIMD
- Programming models
  - Single Program Multiple Data, SPMD
  - Multiple programs Multiple Data, MPMD
- Concurrent, Parallel, Distributed
  - **Concurrent:** A single program with multiple tasks in progress
  - **Parallel:** A single program with multiple task closely cooperating
  - **Distributed:** Several programs (loosely) cooperating

Architecture – SIMD

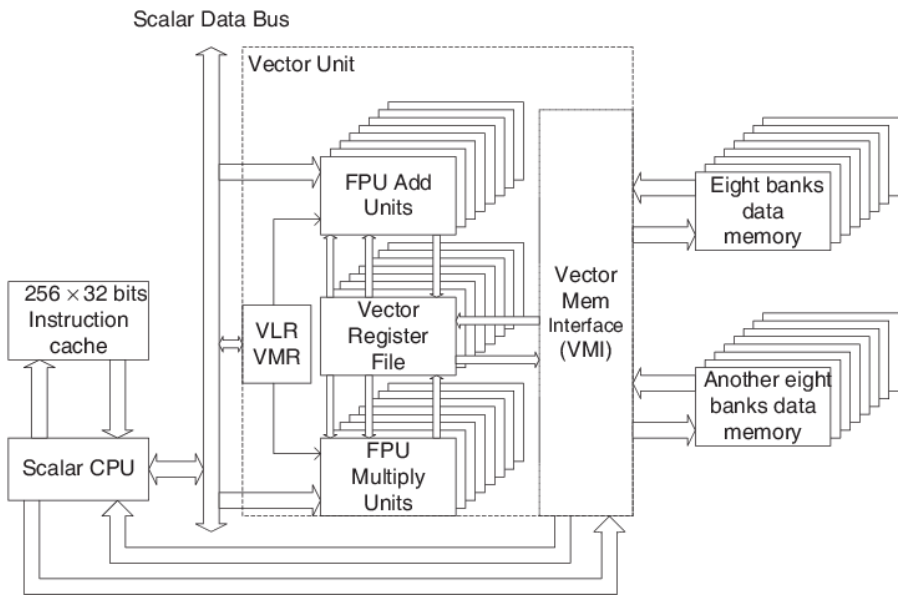
- All processors synchronized
  - All performing the same instruction in any given time
  - Analogy of vector processors
- Simple processors
- Simple programming model
  - but difficult programming (esp. for simple scalar operations)

Vector processor

- Processor able to work directly with *vectors* of data
  - vector is a data type of the underlying instruction set
  - Cray introduced even vector registers (otherwise direct work with the memory)
- Vector Load/Store
  - "composing" vector from different memory words/areas
  - vector of registers that keep addresses of memory words with actual data
  - "to localize" data for further processing
  - in practice the scather/gather operations directly over the main memory

Vector processor

- Memory subsystem
  - as a default does not work with caches
  - interleaved (banked) memory
  - several concurrent operations over the main memory
  - it has higher throughput than use of caches esp. when the data are “randomly” scattered over the main memory (random access to data)



Architecture – MIMD

- Fully asynchronous system
- Individual processors fully independent
  - No special production needed (off-the-shelf)
- Advantages
  - Higher flexibility
  - At least in theory higher efficiency
- Disadvantages
  - Explicit synchronization
  - Difficult programming (easy race condition)

Communication models

- Shared Memory Architecture
- Message passing

Shared Memory Architecture

- Memory separated from processors
- Uniform access to the memory
- Bus as the easiest interconnect
- *Cheap* interprocess/thread communication
- Complex overlap of processing and communication (active waiting)

Message Passing

- Each processor “visible”
- Each processor has its own memory
- Explicit communication – message passing
- High communication cost (exchange of data)
- More easier overlap of processing and communication

Hybrid systems

- Nonuniform memory access architecture (NUMA)
- Cache-only memory access architecture (COMA)
- Distributed shared-memory (DSM)

Non-uniform memory access

- Access to different memory addresses takes different time
- Provides for higher scalability
- Potentially lower throughput
- Cache memory coherence problem
  - ccNUMA – cache coherent NUMA

Cache only memory access

- NUMA but resembling cache memory behavior
- Data moves close to the processors that uses them
- No hierarchy like with caches
  - System must check that the last copy remains (can't delete it)
- Experimental
- Software-based hybrid NUMA-COMA implementation provided by ScaleMP
  - A shared-memory multiprocessor system on top of a cluster of commodity servers

Distributed shared-memory

- Distributed system – cluster
  - Local memory on each node
  - Remote memory on other nodes

  “Fiction” of a single extended memory
- Hardware solution
  - Usually based on virtual memory principles (move pages between nodes)
  - Transparent
- Software solution
  - Library
  - Non-transparent, programmer must modify the program (call of proper APIs from the library)
- ScaleMP hybrid

Cache Memory Coherence

- Reasons for cache miss:
  - **Compulsory** miss: 1st access to data
  - **Capacity** miss: insufficient capacity
  - **Conflict** miss: different memory areas mapped to the same cache row
  - **Coherence** miss: different data in different caches
- Multiprocessors exposed to the last case
  - But it can happen in a single processor case, too (how?)

Invalidation

- Reaction on content change in remote (cache) memory
- Row in the actual (“snooping”) cache memory invalidated
- If the same row is needed later, it is retrieved from the memory (again)

Update

- The cache memory row is updated immediately
- If data are needed (again), they are already in the cache
- Drawbacks
  - False sharing
    - Row includes more words
  - High load on the bus (broadcasting interconnect)
- *Invalidation* and *Update* are equivalent performance-wise
  - Unless a specific memory access pattern is used

Coherence Cache Miss solutions

- Cache memory must be aware of a change elsewhere
- Broadcast based protocols
- Directory based protocols

Snoopy cache

- Broadcast based protocol
  - Communication network with a “natural” broadcast
- Each processor follows/watches *all* memory accesses

Directory based protocols

- Snoopy protocol based on broadcast
  - Not usable for more complex interconnect networks
  - Not scalable
- Solution: Reduction of actively "touched" caches – **Directories**
  - Tag at each memory block
  - Cache memory with a copy of such a block explicitly references the tag
  - Special *exclusivity* tag (writing)

Directory based protocols

- Three based schemas
  - Fully mapped directories
  - Limited directories (partially mapped)
  - Chained directories
- We will compare them based on the following features
  - Size of the additional memory needed
  - Number of necessary instructions/steps (latency introduced)

Fully mapped directories

- Each memory block is able to directly reference all caches (processors) simultaneously
- Bit vector *of copies*
  - If a bit is set, the corresponding cache keeps a copy of the data block
- Exclusivity tag
  - One per a block
  - Writing can be performed on one processor (one cache) only
- Additional tags for each block in each cache
  - Validity tag
  - Exclusivity tag

Limited directories

- Full directories rather expensive (long bit vectors)
  - Additional memory needed: PM/B
    - P number of cache memories (processors)
    - M size of the main memory
    - B block size
- Cumulative capacity of all caches usually smaller that the size of the main memory
- Data block are usually not extensively shared
  - Most directory entries contain zeros
- Solution: Use of direct references
  - However, one bit per cache is not enough

Limited directories II

- Set of pointers to the caches
  - Dynamic allocation as needed
- Features
  - Number of bits per pointer: $\log_2 P$
  - Number of pointers in the pointer pool: k
  - More memory efficient than directly mapped if $k < \frac{P}{\log_2 P}$
- Invalidation information sent only to caches keeping the copy of changed data

Overflow

- If the pointer's pool is exhausted
  - Too many shared blocks
- Possible reactions
  - Invalidation of all shared blocks (broadcast, Dir$_i$B)
  - One entry selection (even random) and invalidation (no broadcast, Dir$_i$NB)

Other modifications

- Coarse-vector (Dir_iCV_r)
  - ***r*** is the size of a region (more caches/processors)je velikost regionu (více procesorů), which corresponds to one bit (i.e. several caches/processors represented by one entry)
    - see the interconnect architectures
  - Switch of interpretation (whether a single processor or a region) as a result of overflow
    - Limited broadcast to all processors in the region
- LimitLESS: software interrupt in case of overflow (“program decides”)

Chained protocols

- Cache-Based linked-list
- Only one pointer per memory block
- Other pointers part of the cache memories
  - The cache memories (their blocks) are “chained”
- Advantages
  - Memory footprint minimization
- Drawbacks
  - Comples protocol
  - More communication (“from cache to cache”; more than theoretically necessary)
  - Write has higher latency (must pass through the whole chain)

Hierarchical directories

- Usable in systems with multiple buses (n general broadcast supporting interconnects)
- Hierarchy of caches
  - Higher hierarchy at each bus interconnect
  - Higher memory requirements
    - Higher hierarchy level must keep copies of shared blocks from lower levels
    - No need of fast ”pointer” memory
- In principle a hierarchy of snoopy protocols with special extensions

Scalability

- No single definition
- Most used definition: Scalable is such a system for which the following holds
  - Performance grows linearly with price
  - The ratio Price/Performance is fixed

  Both are equivalent, but usually only one condition is explicitly used
- Alternative parameter – **Scalability Extent**
  - Performance change as a result of a processor addition
  - Price change as a result of a processor addition
  - Rational extent of number of processors considered
- e.g. the Price/Performance ratio is not constant but within a defined interval (of number processors) the changes are small (and much higher outside the interval)

Speedup

$$
\begin{aligned}
S(N) &= \frac{T_{EXEC}(1)}{T_{EXEC(N)}} \\
&= \frac{T_{comp}(1) + T_{comm}(1)}{T_{comp}(N) + T_{comm}(N)}
\end{aligned}
$$

- Ideal speedup means

$$
\begin{aligned}
T_{comp}(N) &= T_{comp}(1)/N \\
T_{comm}(N) &= T_{comm}(1)/N
\end{aligned}
$$

Speedup – commentary

- Theoretical feature, reality depends on the application
  - Different values for different applications (on the same system)
- Amdahl law – extent of possible parallelization
  - Parallelizable and serial part of the task
  - Parallelizability has its limits

Extensible interconnecting networks

- Parallel systems must include interconnecting network
  - It influences behavior of the parallel system
- Ideal extensible network
  - Low cost growing linearly with the number of processors (***N***)
  - Minimal latency independent of ***N***
  - Throughput grows linearly with ***N***

Network properties

- Three basic components
  - Topology
  - Switching (how data moves between nodes)
  - Routing (how a path is computed)

Interconnecting networks

- We distinguish the following parameters
  - Network size – number of nodes ***N***
  - Node degree ***d*** (number of edges from a node)
  - Network radius ***D***
    - Longest shortest path
  - Bisection width ***B***
  - Network redundancy ***A***
    - Minimal number of links that must be removed for the network to become split into two disconnected parts
  - Cost ***C***
    - Number of links in the network

Bisection width

- Bisection width
  - Minimal number of links that must be removed for the network to split into two same size parts
- **Bisection bandwidth**
  - Commutative throughput of all removed links
- Ideal properties:
  - Bisection bandwidth per process is constant

Topology of interconnecting networks

- Classification based on the number of dimensions
  - One-dimensional
  - Two-dimensional, planar
  - Three-dimensional, cubic
  - Hypercube, tesseract (higher dimensions)

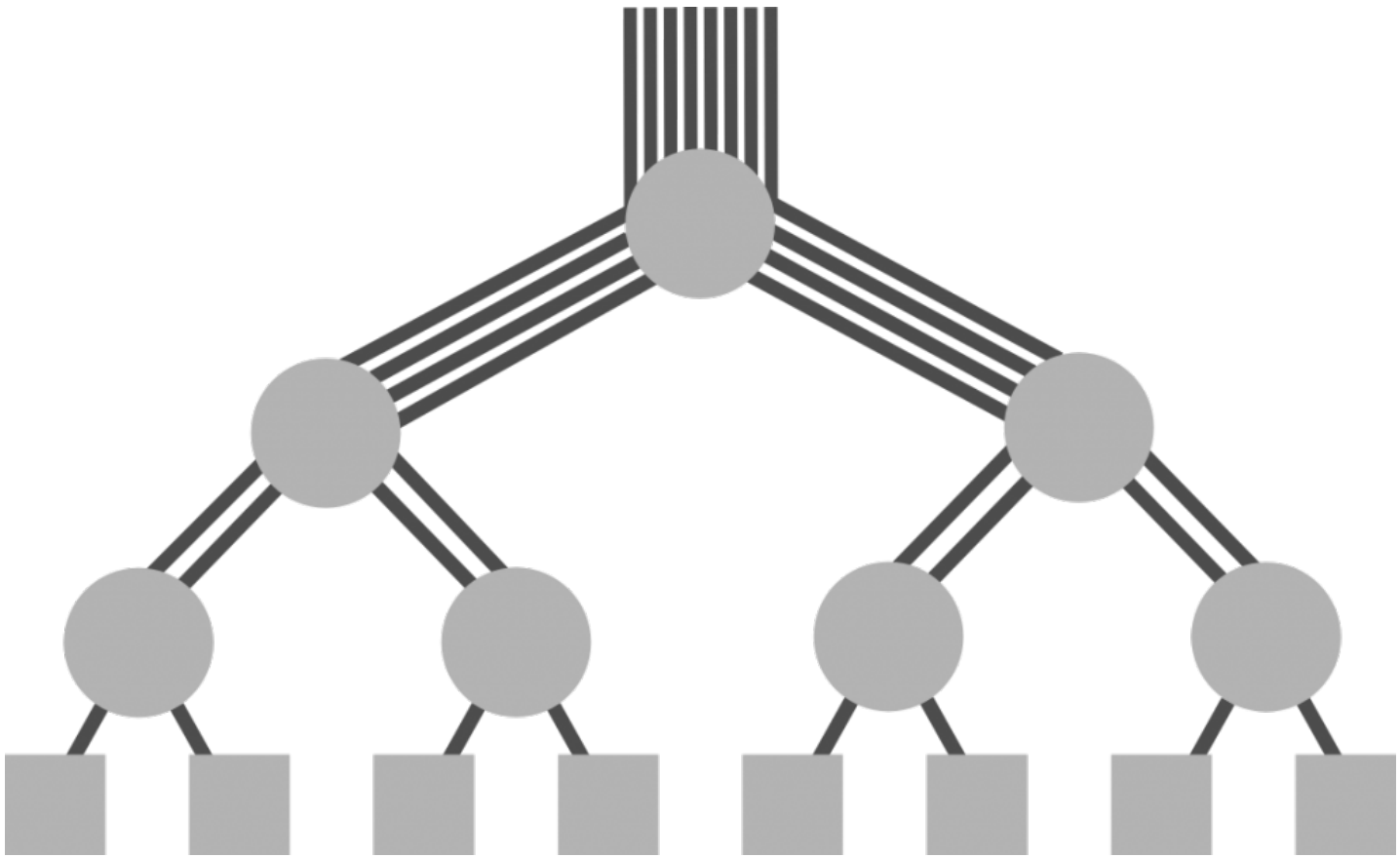
One-dimensional interconnects

- Linear array
- Individual nodes serially connected
  - “(string of) beads”
- Simplest
- Worst properties (for $N > 2$)

Two-dimensional interconnects

- Ring
  - Closed linear array
- Star
- Tree
  - Decreases network radius ($2 \log \frac{N+1}{2}$)
  - Still bad redundancy and bisection (band)width
  - **Fat tree**
    - Adds redundant links at higher levels
    - Improves bisection bandwidth

Fat tree topology



Two-dimensional mesh

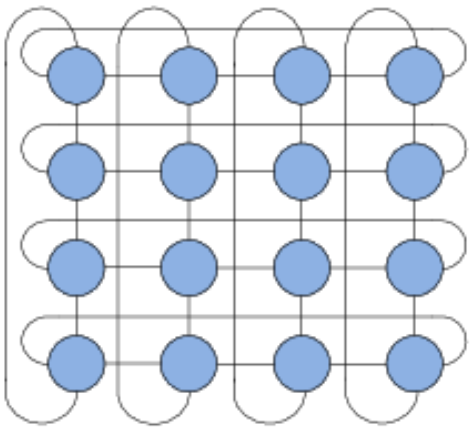
- Very popular
  - Good properties
    - Radius $2(N^{1/2} - 1)$
    - Bisection $N^{1/2}$
    - Redundancy 2
  - However a higher cost and variable node degree
- Torus
  - closed two-dimensional mesh
    - Radius $N^{1/2}$
    - Bisection $2N^{1/2}$
    - Redundancy 4
    - Higher cost – adds $2N^{1/2}$ links

Three-dimensional mesh

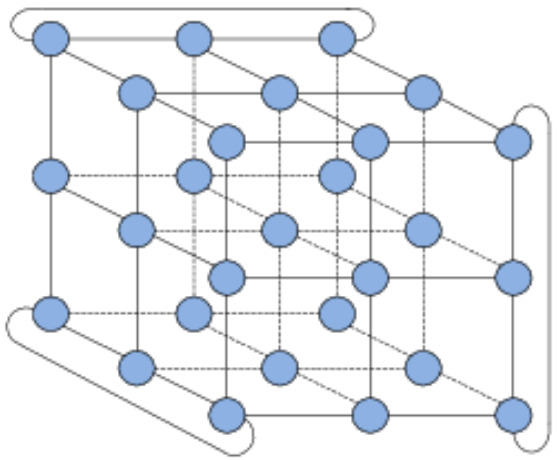
- Properties
  - Radius $3(N^{1/3} - 1)$
  - Bisection $N^{2/3}$
  - Redundancy 3
  - Acceptable cost $2(N - N^{2/3})$
- Difficult for construction

Torus topologies

1-D Torus (4-ary 1-cube)

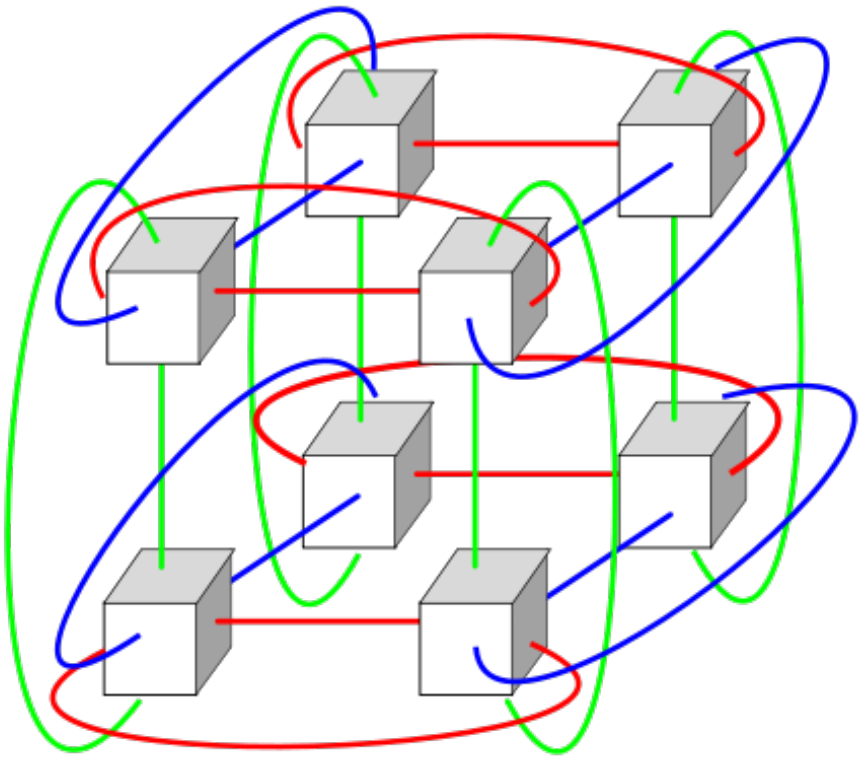


2D Torus (4-ary 2-cube)

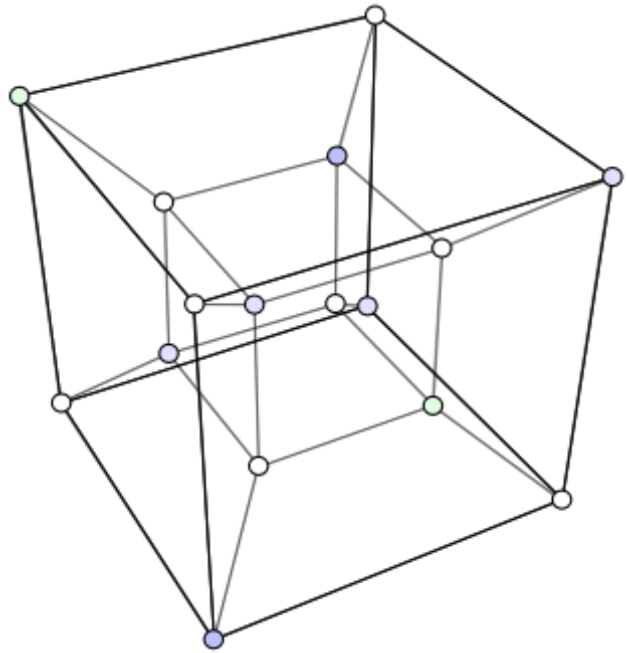


3D Torus (3-ary 3-cube)

Torus topologies

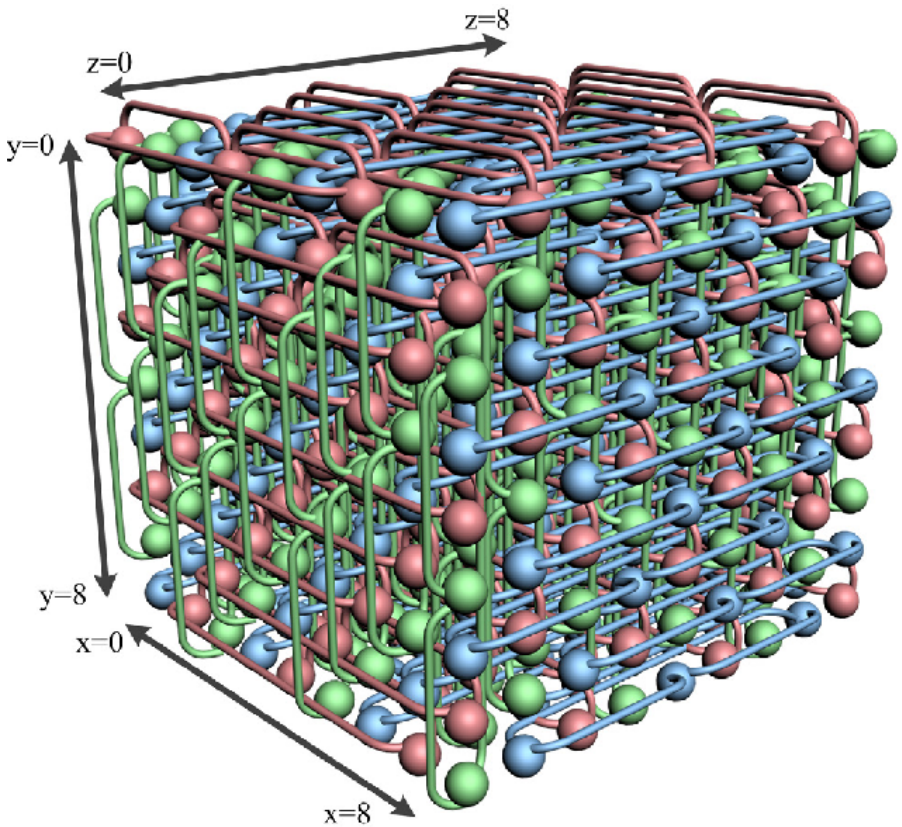


Hypercube, tesseract



- Very interesting topology
- In general n-dimensional cube
- Basic parameters
  - Radius $\log N$
  - Bisection $N/2$
  - Redundancy $\log N$
  - Higher cost $(N \log N)/2$
- Meshes are special cases of hypercube (lower dimensionality)
- Routing very simple
  - Based on binary numbering of nodes

Deadlock Free 6D torus (K Computer)



Fully connected network

- More as a theoretical construct
- Excellent radius (1)
- Unacceptable cost ($N * (N - 1)/2$) and node degree ($N - 1$)

Switching

- Specific mechanism how the packet move from input to output (on a transit node)
- Basic approaches
  - Packet switching, store-and-forward
  - Circuit switching
  - Virtual connection (cut-through)
  - Wormhole routing

Store-and-forward

- The whole packet is stored on the transit node
- And is send out only after the full packet is received
- Simple (first generation of parallel computers)
- High latency $\frac{P}{B} * D$
  - P is packet size, B is throughput and D is number of "hops" (distance)

Circuit switching

- Three stages
  - Connection setup – initiated by a probe
  - Data transmission
  - Connection tearing
- Visibly lower latency $\frac{P}{B} * D + \frac{M}{B}$
  - P is the size of the probe and M is a size of the message (packets are not needed nor relevant)
  - For $P << M$ latency is independent of the path size

Virtual connection

- Message is split into smaller blocks – flow control digits/units (**flits**)
  - The first flit contains the path info (initially the target address)
  - Next flits contain the data (payload)
  - Last flit breaks the path
- Individual flits are sent as a continuous stream
  - With buffers of sufficient size, this responds to the *circuit switching*
- Latency $\frac{HF}{B} * D + \frac{M}{B}$
  - HF is flit length, usually $HF << M$

Wormhole

- Special case of virtual circuit
- Buffer size exactly fits the flit size
- Latency does not depend on the distance (the length of the path)
  - Pipeline analogue
  - The whole packet resides in several buffers at different nodes on the path – thus the *wormhole*
  - Latency is considered at the level of the whole message transfer, not per flits; number of flits is much higher than the distance (the length of the path = number of hops)
    - transmission time is a **sum** of the length of the path and number of the transferred flits
    - while for store-and-forward it relates to the **product** of these two values
- The protocol supports packet replication
  - convenient for multicast and broadcast

Wormhole Three Interfering Flows

Virtual channels

- Physical channels sharing
- Several buffers over the same channel
  - Flits stored in the appropriate buffer
- Use
  - Overloaded links
  - Deadlock avoidance
  - Logical to physical topology mapping
  - Guarantees of sufficient transport capacity for system data

Routing in the interconnecting networks

- Path discovery
- Properties
  - Static routing
    - Source based
    - Distributed
  - Adaptive routing (always distributed)
- Minimal but also non-minimal routes

Fault tolerance of interconnecting networks

- Error check
- Message acknowledgment
- Message re-transmission

Memory latency

- Memory considerably slower than the processor
  - Waiting for memory substantially decreases efficiency of the whole system
- Solutions:
- *Latency decrease* – access speedup
- *Latency hiding* – interleaving of computing and data transmission

Memory Latency Decrease

- **NUMA:** Non-Uniform Memory Access
  - Each logical address is mapped to a concrete physical address
- **COMA:** Cache-Only Memory Architecture
  - Main memory considered as an *attraction memory*
  - Memory rows can be moved freely
  - A memory row can have several copies

Recapitulation

Communication to computation ratio	NUMA		COMA	
	Small working set	Large working set	Small working set	Large working set
Low	Good	Medium	Good	Good
High	Medium	Poor	Poor	Poor

Memory Latency Hiding

- Weak consistency models
- Prefetch
- Multiple-context processors
- Producer initiated communication

Weak consistency

- Does not require a strict synchronization access to shared variables unless explicitly required (explicit synchronization)
- **Release consistency:**
  - New instructions *acquire* and *release*
- *Fence* instruction
  - Enforced finalization of unfinished instructions (waiting)

Prefetch

- Data moved to the process in advance
  - *Binding* prefetch
    - Data moved to the processor (register)
    - Risk of consistency break
  - *Non-binding* prefetch
    - Data moved to the cache only
- HW Prefetch
- SW Prefetch
- Special instruction *prefetch-exclusive*: read followed by a write
- Remember ANDES?

Multiple-context processors

- Multithreading support
- Requires
  - Very fast context switch (1-2 ticks)
  - Very high number of registers (the full set per each thread)
- Many experimental systems
  - HEP (seventies)
  - Tera
  - *T

Producer initiated communication

- Analogy of invalidate and update in cache coherency
- Specific use for message passing systems (e.g. Cray T3D) or block-copy (computers with shared memory)
- Suitable for transfer of large blocks or for lock-based synchronization

Synchronization support

- Synchronization leads to “hot spots”
- Basic synchronization primitives/protocols:
  - Mutual exclusion (mutex)
  - Dynamic load distribution
  - Events’ propagation
  - Global serialization (barriers)

Mutual exclusion (mutex)

- Access to a shared variable is granted to at most one process at any given time
- Universal, but usually rather expensive
- Synchronization constructs of higher programming languages
  - Semaphores
  - Monitors
  - Critical sections
- Foundation – hardware support
  - `test&set` instruction
  - `test-and-test&set` instruction

  Spin waiting protocol

test&set

- Properties
  - An atomic instruction that reads the actual content and sets the content to 1 (CLOSED)
  - Busy (active) waiting till the read value is 0

    ```
    char *lock;
    while (exchange(lock, CLOSED) == CLOSED );
    ```
- Highly stressing cache coherent multiprocessor systems
  - Each “set” flushes all caches

test-and-test&set

- Properties
  - An instruction that reads the actual value and run the atomic test&set only if the read value is 0

    ```
    for (;;)
    while (*lock == CLOSED);
    if (exchange(lock, CLOSED) != CLOSED)
    break;
    ```
- Protocol responsive to the cache subsystem properties
  - First test over the shared (in cache) copy

Use of queues

- Improvement: *Collision avoidance schemes*
- Queue on lock bit (QOLB) protocol
- The most effective implementation
- Processes lined up in a queue
  - After lock release the process at the queue head is activated
  - No data transfer among all waiting processes needed

Locks in multiprocessors

- Related to the dynamic load distribution
- Use of counter with atomic operation
  - Fetch&Op – counters, e.g., Op==Add

```
fetch&add ( x, a)
int *x, a;
{ int temp;
  temp = *x;
  *x += a;
  return (temp);
}
```

  - Compare&Swap – lists

Event's propagation

Global events/signals used as a tool

- for producers to inform consumers that data are ready
- to inform about a global change in a set of equivalent processes
  - Status change that must be announced to all processes

Barriers

- A point of synchronization
  - To let all processes synchronize at the same point
- No process can pass the barrier unless all processes reached it